Pers Ubiquit Comput (2016) 20:185–194
DOI 10.1007/s00779-016-0903-2

ORIGINAL ARTICLE

Declarative GUI descriptions for device-independent applications

Jacek Chmielewski[1] · Jakub Flotyński[1] · Dariusz Rumiński[1] · Adam Wójtowicz[1]

Received: 6 April 2015 / Accepted: 5 January 2016 / Published online: 2 February 2016

Abstract The increasing number and diversity of devices connected to the Internet open new research challenges in the field of cross-platform and device-independent applications. One of the approaches to this problem is the Device-Independent Architecture, which provides application logic and application data device independence. It enables also usage of user interface (UI) adaptation middleware to support application UI device independence. Potentially, device-independent descriptions of application UI can be implemented with existing user interface description languages (UIDLs). In this paper, we present an analysis of eight popular UIDLs that are assumed to be suitable for device-independent GUI descriptions, along with a summary of evaluation results and lessons learned. The selected UIDLs were employed to describe a set of GUI views based on an existing mobile application. The gathered results confirm our research hypothesis that the analyzed popular declarative UIDLs are not capable of describing mobile GUI in a device-independent manner. Therefore, using the knowledge gathered from the reported experiment, we propose a set of guidelines for an optimal device-independent UIDL.

✉ Jacek Chmielewski
chmielewski@kti.ue.poznan.pl

Jakub Flotyński
flotynski@kti.ue.poznan.pl

Dariusz Rumiński
ruminski@kti.ue.poznan.pl

Adam Wójtowicz
awojtow@kti.ue.poznan.pl

[1] Department of Information Technology, Poznań University of Economics and Business, Al. Niepodległości 10, 61-875 Poznan, Poland

Keywords User interface description languages · UIDL · GUI adaptation middleware · Device-independent applications · Device-Independent Architecture

1 Introduction

Cisco's Internet Business Solutions Group predicts that by 2020 there will be about 50 billion devices connected to the Internet [8]. There are already more connected devices than people on the planet. But, according to the mentioned forecast, by 2020 there will be almost ten devices per each human being. The rapid growth in the number of devices will be followed by tenfold increase in mobile data traffic [9]. This trend builds the need for solutions that enable users to access their data and applications in a device-agnostic way—i.e., no matter what device type or software platform they use. Solutions for accessing distributed data sources using the ubiquitous Web platform were proposed in [30, 23]. But, to provide fully device-independent applications it is necessary to overcome three major obstacles: device independence of application logic, device independence of application data, and device independence of application user interface (UI). Researchers usually approach these issues separately, but there are solutions that have the ability to address them all at once. One of the available options is to build applications according to the Device-Independent Architecture (DIA) [6, 7]. The DIA makes application logic and data device independent by placing them outside of user's devices. The only element of device-independent applications that depends on device parameters is the UI. To maintain complete device independence of an application, it is necessary to introduce a UI adaptation middleware between the application and user's devices. In such configuration, the application has to

provide a device-independent UI description, which can be processed by the middleware and adapted for a particular device. Adaptation of UI to different target devices can significantly reduce effort in application development—in comparison with creating independent UIs for the particular devices [20]. In addition, to maintain application-middleware independence, the device-independent UI description should not depend on a particular middleware implementation. This goal can be achieved by providing the middleware with declarative description of an application UI, as contrary to imperative instructions defining how to generate the UI within a specific programming platform.

Assuming the usage of a DIA-based application and a UI adaptation middleware the question that remains is: How to construct a device-independent UI description? Answering the question requires solving several diverse problems that depend on required UI modality. The scope of the research presented in this paper is limited to the most common UI modality: the graphical UI (GUI). Considering the GUI, the problems that have to be solved are determined by two major tasks composing a GUI adaptation process: adaptation of GUI structure and adaptation of GUI appearance. The first task is about deciding which GUI elements should be presented on a particular device. The second task is to provide information on how these GUI elements should look like. The work presented in this paper addresses problems related to the second task. Our research hypothesis is stated as follows: the existing popular declarative UI description languages (UIDLs) are not capable of describing mobile GUI views in a device-independent manner for DIA-based applications. To test this hypothesis, we have defined a set of GUI views based on an existing mobile banking application that has multiple versions for different devices (different software platforms) and we have conducted a series of experiments aiming to describe the defined GUI views using eight popular UIDLs. Based on the results of these experiments, our main contributions presented in this paper include the following:

- confirmation of the hypothesis that none of the analyzed UIDLs are able to fully support device-independent GUI descriptions for DIA-based applications,
- formulation of a set of practical guidelines that could be used to improve some characteristics of existing UIDLs or to develop a new device-independent UIDL.

Both contributions push the research on DIA-based device-independent applications forward and open new research directions that could be investigated in this area.

The rest of this paper is constructed in the following manner. Section 2 provides background information on DIA and UIDLs, along with an overview of similar research conducted in the past. Sections 3 and 4 present the experiment design and provide a description of experiment results. Section 5 contains a discussion and an analysis of the results. The final conclusions and summary are provided in Sect. 6.

2 Background

The background of the presented survey covers the DIA, which separates applications from end-devices, and UIDLs that enable device-independent UI description, which can be used within applications based on the DIA.

2.1 Device-Independent Architecture

The DIA has been proposed to facilitate analysis and development of applications that can be made available to users via any capable device from the large, diverse, and fast-growing pool of Internet-enabled end-devices—i.e., devices that are used directly by users to interact with an application, but not sensors that passively record a state of an environment. As presented in [6], the idea of DIA originates from the service-oriented architecture, where systems are decomposed into atomic services, and processes use such services without knowing much about their implementation. A similar approach can be used to decompose end-devices. Each end-device, be it a laptop or a smartphone, provides: resources, services, and user interaction channels. Resources encompass processing power, memory, and storage. Services are providers of context information, such as location, temperature, light intensity, and data from other types of sensors. User interaction channels (both incoming and outgoing) are the means to communicate with a user and include screen, keyboard, vibration, and camera. The key concept is to use external resources, instead of what is provided by an end-device, and to generalize the way services and user interaction channels are accessed. Therefore, in DIA, the separation of application from end-devices, which enables the device independence, is achieved by:

- executing the application outside of end-devices,
- accessing sensor data provided by a device via a standardized API, and
- using universal UI descriptions.

The execution of the application on external resources ensures that the application logic does not depend on the hardware or software platform of an end-device. The interesting consequence is that, in this architecture, end-devices could be deprived of their general purpose resources, as the resources are not needed. Services publish data in service-specific formats (e.g., location coordinates

for a geolocation service, numerical data for a temperature sensor) independently of their implementation on a particular end-device. Therefore, it is feasible to build a middleware providing a device-independent API, such as the one proposed in Wolfram Language [31], to access such services. The usage of a universal UI description is a key requirement for independence of an application UI from end-device parameters (e.g., screen size and pixel density).

However, such generalized UI descriptions may result in a quite raw UI presented to users, which is usually far from what is expected for a consumer mobile application. To enable UI presentation to be tailored to parameters of a specific end-device, the generic UI description has to be properly adapted before reaching the user. UI adaptation should be separated from the application and could be provided in the mentioned middleware that is placed between applications and end-devices. The structure of an application implemented according to DIA is presented in Fig. 1. The middleware functionality may be bound to the application itself as a software library, may be provided as a separate proxy service, or may be implemented in a form of a *device independence driver* directly on an end-device.

The research presented in this paper focuses on the flow of GUI description—since its generation at the server side, to its presentation on an end-device. To maintain the overall device independence of an application, the GUI description generated by the application executed at the server side has to be device independent. Then, the middleware uses information about end-device screen parameters to adapt the GUI description to its final form, which is subsequently passed to the end-device for presentation.

2.2 User interface description languages

As defined in [12], a user interface description language (UIDL) is a specification language that describes various aspects of a UI under development. UIDL defines a syntax and semantics that can be used to support the various stages of UI development life cycle and development goals, and determines the granularity of UI components that can be used in UI design [29]. Recently, the most notable research in the area of UIDLs was done by the W3C model-based UI Working Group (MBUI WG) [17]. Its goal was to combine the best elements from existing UI description solutions and to propose recommendations that will enable building context-aware UIs for a variety of Web interactive applications. The MBUI WG indicates the CAMELEON Framework, presented in Fig. 2, as the main reference for classifying UIs supporting multiple targets—users, platforms, and environments—in context-aware computing. The framework defines development of UIs at the following four levels of abstraction. From the highest level—the *task and concept* level—on which logical activities, which are required to reach users' goals, and domain specific objects, which are manipulated by these activities, are specified at design time. To lower abstraction levels, which are used at run time. Following the concepts provided by the CAMELEON framework, in this research we assume the following UI levels.

- At the *abstract user interface* (AUI) level, a UI is represented as a collection of presentation units, which are abstract in the sense of their final presentation (independent of the modality of interaction), as they can be presented (accessed) in different ways, e.g., visually, vocally, haptically. Presentation units typically focus on semantics and general properties of UI components (e.g., input / output data components as well as structural and logical dependencies between components), without covering exact properties related to the final presentation of the UI (e.g., size, position, color of components).
- At the *concrete user interface* (CUI) level, a particular modality is selected and a number of additional descriptive attributes are introduced to an AUI description, which has been created at AUI level, to describe the UI more precisely and to enable the perception of a UI by a user independently from a particular presentation platform (e.g., layouts, relative size and position of components).
- At the *final user interface* (FUI) level, a CUI description, which has been created at CUI level, is encoded in a particular UI description language (a programming language or a mark up language, e.g., Java, HTML5, VoiceXML). A FUI is typically specific to a selected modality of interaction as well as particular hardware and software platforms (devices, operating systems, presentation tools), as it may specify, e.g., properties depending on screen resolution, or type of keyboard. A FUI description may be either compiled or interpreted. It may be presented in various forms, on various platforms depending on, e.g., device capabilities, browser implementation, or the context of interaction. In general, the border between CUI and FUI descriptions may be fuzzy, as some FUI level UIDLs may be used to represent UIs at CUI level, by skipping UI

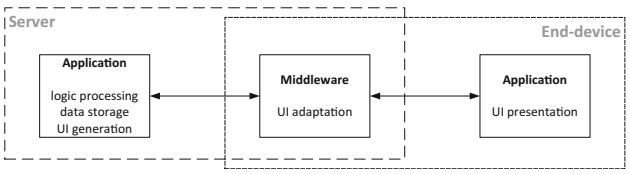

Fig. 1 Device-Independent Architecture diagram

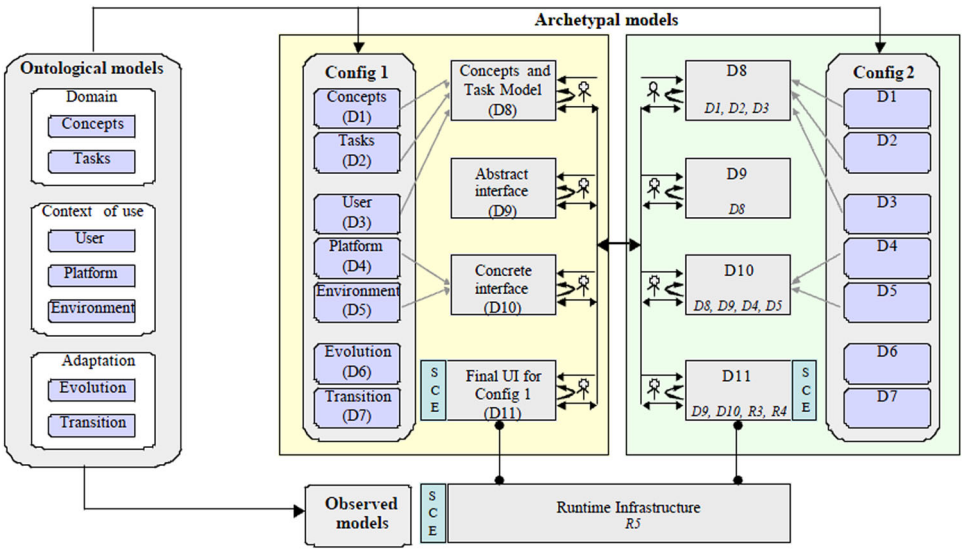

Fig. 2 CAMELEON reference framework (image taken from [5])

components and properties that are directly related to particular hardware or software platforms and using only components and properties that are platform-agnostic.

The aforementioned levels can provide the basis for modeling UIs according to the task-oriented paradigm. The main activities leading to the creation of a UI, temporal relations between the activities as well as attributes of the activities and manipulated objects are described in [18, 19].

A few surveys have been conducted to compare capabilities of different UIDLs in the context of building multi-platform UI descriptions. In [25], selected AUI level UIDLs have been compared in terms of expressiveness, available concepts, openness, and the number of tags. In [12], various AUI and FUI level UIDLs have been compared in terms of various aspects such as popularity, methodology used, available tools, supported platforms, and tags. Only languages that are relatively well documented, available for testing, and used in some development cases have been selected for the survey. In [28], general requirements (e.g., target device, delivery context, possibilities of personalization and extensibility) and technical requirements (e.g., separation of an interface from its presentation, run time, and remote control) have been specified for AUI level UIDLs, and selected languages have been evaluated. In [24], selected CUI and FUI level UIDLs have been compared in terms of available tools, supported target platforms, description of styles, support for vector graphics, and the number of tags. Moreover, a case study has been considered, and example applications have been developed using the analyzed languages. The comparison of the developed applications allowed for more comprehensive evaluation of the languages. In [22], several well-established and well-documented languages have been compared in terms of Web application development. In particular, the comparison includes criteria related to: device and modality independence, separation of data and interface, capabilities of UI components as well as remote, and real-time control.

The aforementioned works have evaluated various UIDLs according to a number of criteria, which are mainly related to expressiveness and possibilities of building multi-platform UIs. However, they do not provide a comprehensive evaluation of UIDLs in terms of their suitability for building device-independent GUI descriptions of DIA-based applications. In these applications, the UIDL has to provide a universal device-independent GUI description that contains all UI presentation details necessary for automatic GUI adaptation to capabilities of a particular end-device. This aspect was not covered in the previous research, and our work aims to fill this gap. Additionally, we define the GUI view to have a wider scope than usual. In our research, the GUI view is an entity composed of GUI components visible on a screen (output) and all possible user actions that trigger a reaction (input). What is important, the input part includes actions that influence the GUI, but are not triggered by any visible GUI component. These actions may occur outside of the screen presenting the GUI, (e.g., a usage of a hardware *back* or *settings* button).

3 Experiment design

The goal of the performed analysis was to verify the hypothesis that popular declarative UIDLs are not capable of describing mobile GUI views in a device-independent manner for DIA-based applications, where the GUI

description can be unambiguously and automatically processed by a GUI adaptation middleware. To test this hypothesis, we have defined a set of GUI views that mimic a selected multi-platform m-banking application [4], we have described these GUI views with eight popular UIDLs, and finally, we have assessed the results using a set of measurable qualitative and quantitative assessment criteria. The m-banking domain has been chosen because it covers a broad spectrum of practical usability requirements: on the level of separate GUI components (e.g., custom password field, specific validation requirements), on the level of complete GUI views (e.g., interaction between GUI components), and on the level of step-by-step usage scenarios composed of many GUI views. The GUI views have been described in all analyzed UIDLs, each in two versions: a device independent (before adaptation) and a device specific (after adaptation). However, in this experiment we focus only on device-independent descriptions.

The experiment was preceded by the initial phase that included:

- defining experimental GUI views and their basic components,
- identifying UIDLs potentially suitable for describing device-independent GUI views,
- specifying assessment criteria for the selected UIDLs.

The three elements of the initial phase have been explained below.

3.1 Experimental GUI views

One of the assumptions of this evaluation was the usefulness of its outcomes in the process of designing a stateful GUI adaptation system for DIA-based mobile applications. Therefore, GUI components typical for stateful interactions, such as in scenarios of HTTP-based request/response, or typical mobile application interactions, have been chosen to be analyzed. For this evaluation, ten different composed GUI views have been specified in a language-independent manner, i.e., *Welcome UI, Login UI, Main menu UI, Details UI, History UI, Transaction details UI, Money transfer form UI, Help UI, Map UI, Exchange rates UI* (sample GUI view is presented in Fig. 3).

Composed GUI views are constructed using a set of proposed universal GUI components. The list of these GUI components includes: *Menu, Details, List, Information form field, Text input form field, Amount input form field with validation, Date input form field with validation, Select form field, Checkbox form field, Password input form field, Form action button, Table element, Map element, Page element, Navigation menu,* and *Banner*.

The selection of GUI components and composed GUI views has been guided by UIs of the m-banking application. The application is available on five different mobile platforms (Android, iOS, Windows Phone, Symbian, and mobile Web), and the analysis included all these variants of the application UI. The goal was to cover a wide range of typical and custom GUI components used in standard and custom layouts. The final set of selected GUI components and experimental views is a superset of what can be found in majority of existing business and productivity applications, in which the user interface is designed as a set of separate views.

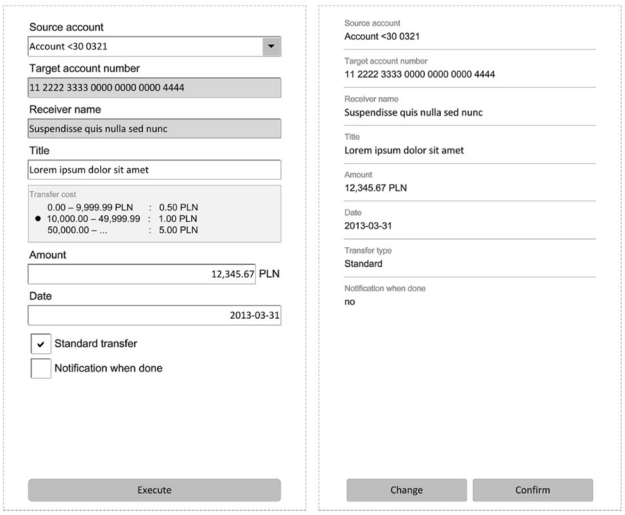

Fig. 3 Example of composed GUI view—*money transfer form UI*

3.2 UIDLs selection

The research presented in this paper is focused on UI descriptions that can be placed between CUI and FUI levels. Initially, we have analyzed also the AUI level UIDLs, such as UIML [1], UsiXML [15], and MARIA [21], which have been intended for device-independent UI description. However, in this research we are looking for a solution that provides UI designers with an extensive UI appearance control on the level of the device-independent UI description—which is not the case for abstract UI languages.

The analyzed device-independent UI descriptions assume fixed modality (visual output, plus touchscreen-based input) and provide all details required for proper UI presentation on a given device—which maps to the FUI level. At the same time, these GUI descriptions have to be device agnostic, so they correspond to the CUI level and may employ only screen independent description constructs (e.g., % and mm units instead of pixels). Therefore, UIDLs used in this analysis have been selected to represent both categories of declarative UIDLs, i.e., low-level (allowing for expressing UI representations that can be

directly instantiated and presented, thus allowing for expressing explicitly, e.g., the positioning and sizing values) and universal languages (integrating the aforementioned functionality, allowing for defining positioning and sizing, but expressed as relative values, independent from particular screen).

Also, expressive power of a UIDL is an important factor for its assessment. A UIDL supporting only predefined high-level GUI components will not be able to meet requirements of mobile business applications, which often use custom GUI components. Therefore, the expectation is that a valuable device-independent UIDL will satisfy two seemingly contradictory requirements: high-level device-independent GUI specification, with an expressiveness of low-level pixel-based custom GUI designs.

The presented analysis is based on eight representatives of declarative UIDLs, which include:

- UIDLs related to major technology vendors: AndroidXML from Google [2], XAML from Microsoft [11], UIBinder from Google [26], MXML from Adobe [10]),
- UIDLs related to popular open-source technologies: QML [27], XUL [16], OpenLaszlo [3],
- the Web-based de-facto standard: HTML [13].

Additional criteria for including the given UIDL in the analysis have been: popularity, usage in significant products, the presence of an active developers community, and availability of a detailed language documentation. Notice that this research is focused solely on declarative GUI descriptions, excluding dynamic aspects based on programming languages bound with GUI descriptions.

3.3 Assessment criteria

The UIDL assessment criteria have been divided into a group of expressiveness criteria (is it possible to express a given GUI component having required features) and a group of clarity criteria (how unambiguous and compact the description is). There are also two additional criteria that are not related to expressiveness and clarity. All these criteria are explained in detail below.

The expressiveness criteria checks whether it is possible to fully express a given GUI element using a selected UIDL. This set of criteria is based on *GUI components* (16 components + 2 additional features) as well as on complete *composed GUI views* (10 interfaces). If a given expressiveness criterion is fulfilled by a given UIDL, one point is added to the total assessment value for the UIDL. If a given UIDL only partially allows for expressing a given GUI component, fractions of the point are added—proportionally to the amount of the missing features. For instance, 0.2 point is counted if *Date input form field* is absent and has to be replaced with a regular text field that preserves its function, but at the same time eliminates the ability to: control the character input process, validate data, or use pop-up calendar. On the other hand, 0.9 point is counted if *Date input form field* allows for controlling value types and value ranges, but forces user to input the date according to one fixed format and there is no way to redefine it within the UIDL.

The clarity criteria verify the UIDL ease of use for developers (less complex structure with smaller number of distinct elements is easier to develop and maintain) and measure the volume and complexity of GUI descriptions. For complexity, the structured document complexity metric (SDCM) has been used. It is a metric that can be applied to documents, or a set of documents, that are represented in both XML-based or SGML-based languages. It provides a single value which is calculated by summing up the number of unique XML elements and the number of unique XML attributes. Moreover, according to SDCM, additional points are added: one for each required element and one for each required attribute. Since only documents, as opposed to document schemas, have been analyzed in our work, it is checked whether given element/attribute is always present in its parent element to test whether it is required or not. For the same reason, the last SDCM rule (additional point added for each element that can only appear as a first child of its parent) is not applied to our calculations. Initially, it was also considered to take into account extra (non-SDCM) complexity factors, such as the presence of recursion or the number of unique namespaces used. However, they have not been used as not significant for the assumed DIA processing model. Finally, volumes of the GUI descriptions (document sizes) have been measured, taking advantage of the fact that all analyzed documents use the same encoding and the results are mutually comparable.

The two additional criteria check whether it is possible to attach user interaction handlers (e.g., onClick) to UI elements (allowing to connect UI with application logic) and whether it is possible to compose multilingual UI descriptions. Each criterion was assigned 1 point or less, in the case of a partial support.

These measures have been used for each GUI component and separately for each *composed GUI* (multiple GUI components combined into a single interface).

4 Results of the experiment

The results of the expressiveness assessment (indicating how many of the required GUI components can be expressed) are presented in Fig. 4 (bars filled with oblique pattern). There is no UIDL with 100 % expressiveness, that

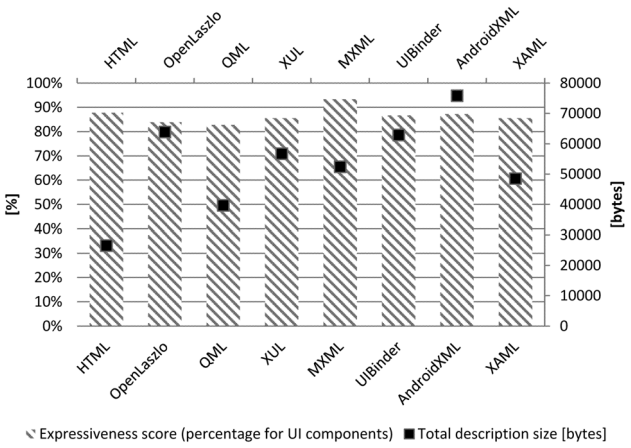

Fig. 4 UIDLs expressiveness score set together with total description sizes

is allowing to express every GUI component specified in the initial phase.

In Fig. 4, the expressiveness of the selected UIDLs is presented together with total description sizes that are summed up for all *composed GUI views* (represented by black squares).

The size in bytes is not an accurate measure of description clarity. There are UIDLs, e.g., HTML, that produce relatively small, compact descriptions, but require large number of distinct, unique elements (tags). Therefore, we have used SDCM instead of relying only on the GUI description size (c.f. Sect. 3.3). A strong negative correlation (0.8) between size (measured in bytes) and complexity (measured in SDCM) is observed, which can be explained by the fact that compact descriptions need complex dictionaries and vice versa.

Figure 5 presents SDCM values for all *composed GUI views* bundled together (bars filled with vertical lines). In this case, the element/attribute uniqueness used to calculate SDCM values is aggregated for all documents globally.

The evaluation algorithm (c.f. Sect. 3.3) has a hidden assumption that each GUI component has equal weight—no matter how significant it is for implementation practice, i.e., whether it appears in many *composed GUI views* or not. To take this significance into account, we have also measured the UIDL expressiveness and SDCM for each composed GUI. Regarding SDCM, contrary to measuring global element/attribute uniqueness that can be perceived as a complexity measure of a UIDL itself, measuring element/attribute average uniqueness within independent *composed GUI views* expresses practical complexity of GUI descriptions developed with a given UIDL. The global element/attribute uniqueness is depicted in Fig. 5 as bars filled with vertical lines, while the element/attribute average uniqueness within independent *composed GUI views* is depicted as bars filled with checkered pattern.

The expressiveness results obtained for independent *composed GUI views* depicted in Fig. 6 are generally consistent with the previous results (c.f. Fig. 4)—there is no UIDL that is able to express all features of every composed GUI from the assumed reference set. Majority of analyzed UIDLs do not support such components as *Map element*, or *Date input form field with validation*, nor do they allow for building a multilingual description, which makes describing fully functional *composed GUI views* (applied to m-banking applications or applications in other professional domains) practically impossible.

On the basis of the gathered evaluation results, it can be stated that MXML, AndroidXML, and OpenLaszlo are the most expressive UIDLs. MXML has the highest scores in two out of three measurements (expressiveness percentage of all UI components and expressiveness as a number of fully implemented *composed GUI views*), AndroidXML—

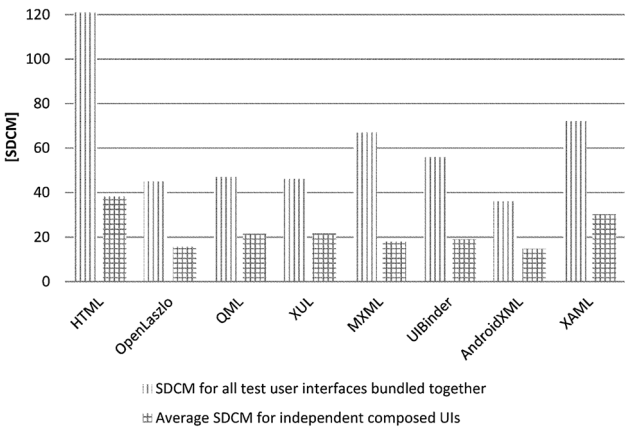

Fig. 5 SDCM values for all GUI descriptions bundled together (*bars* filled with *vertical lines*) and average SDCM values for individual *composed GUI views* (*bars* filled with *checkered pattern*)

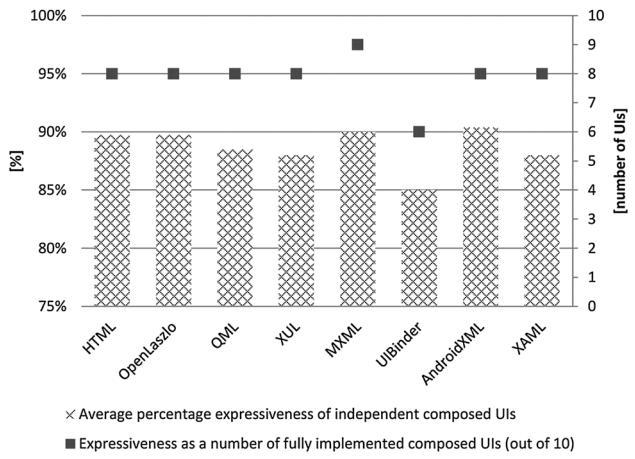

Fig. 6 Average percentage expressiveness of independent *composed GUI views* and expressiveness as a number of fully implemented *composed GUI views* (out of 10)

in one (average percentage expressiveness of independent*composed GUI views*). In terms of clarity, Android XML is the best language, OpenLaszlo is good, while MXML is average. Generally, the differences between UIDLs are observable, but not large, except for UIBinder, which is definitely the least expressive UIDL (probably since it is designed to be used in conjunction with imperative Java GWT code), and for HTML, which has produced compact but complex descriptions composed of many specific elements.

5 Discussion

As it has been illustrated in the previous section, none of the analyzed UIDLs enable full implementation of the selected GUI views and of device-independent GUI views for DIA-based applications in general. Declarative GUI descriptions that are to be useful in the context of the DIA should take into account three aspects: total number of unique GUI elements provided by an UIDL, GUI element appearance specification, and GUI behavior specification.

The DIA is based on the assumption that end-devices can be thin clients—i.e., devices without or with little processing power. Therefore, the interpretation of a GUI description, necessary for GUI presentation on a device, should be kept as simple as possible. In consequence, the total number of unique GUI elements provided by an UIDL should be as low as possible. More GUI elements mean more complex GUI interpretation (e.g., HTML), but less GUI elements will constrain the GUI design (e.g., AndroidXML). Potential solution is to follow the approach adopted by graphic APIs (e.g., DirectX, OpenGL) and compose GUI views using only graphic primitives (text, line, rectangle, etc.). This would limit the number of unique GUI element and simplify the GUI interpretation on end-devices, but it would also have negative consequences. It would result in bigger (multiple elements required to express GUI constructs) and potentially complex GUI descriptions (multiple interrelated GUI elements). Therefore, the balance between the GUI interpretation complexity and the GUI design complexity should be investigated in the context of a specific GUI adaptation method.

The second aspect of GUI description is the appearance of GUI elements (i.e., element position, size, color, and other formatting options). The majority of analyzed UIDLs address this issue very well by providing formatting tools based on the popular CSS specification. Only UIDLs tightly related to programming environments have minor formatting limitations, as they are designed to be used in conjunction with imperative code that influences the GUI formatting.

The third aspect of GUI description is related to GUI behavior. Since the aim is to provide a GUI description for an application used directly by users, it is necessary to specify how to handle user input—i.e., what user actions should be caught and what should happen in response. Most of the analyzed UIDLs allow for describing what to show to a user, but not how to interact with a user, especially, when such interactions may be triggered by events that happen outside of the GUI (e.g., the *back* functionality is usually triggered by a physical *back* button). Missing features include also the ability to express interactive GUI components (i.e., components that should change directly in response to some user actions, e.g., *Interactive information form field, Map*) and validation rules for input GUI components (*Amount, Date*). Within DIA model, all user actions could potentially be redirected to an application on the server side and initiate generation of appropriate GUI updates. But it would introduce additional delay in the application response time and would make the application GUI susceptible to connectivity problems. So it would be practical to handle GUI related interactions directly on an end-device.

Based on the specificity of the DIA, the results of the presented experiment, and lessons learned during the implementation of the experimental GUI views, it is possible to formulate a set of guidelines that could be used to improve some characteristics of the existing UIDLs or develop a new device-independent UIDL. The optimal device-independent UIDL should be able to provide all details necessary for automatic GUI adaptation for DIA-based applications, without constraining UIDL expressiveness available to a GUI designer. Such optimal UIDL should follow the following guidelines:

- To maintain independence of device display pixel density, all positioning and sizing measurements of GUI components should be expressed only using device-independent values (percentage) and units (mm, pt, etc.). This guideline results from device independence requirements imposed by DIA.
- To inform the end-device about how to handle user actions, every GUI component should enable specification of user interaction handlers and the UIDL should allow for attaching user interaction handlers to external triggers (e.g., physical buttons). DIA-based applications use the *input redirection* [14] paradigm, so to optimize the client–server communication it is necessary to specify which user actions should trigger user input events redirected to the application on the server side. Moreover, to preserve device independence of the UIDL in the context of future end-devices, the dictionary of supported user interactions (UI handlers) should be extensible.

- Another optimization axis is the volume of data transferred between a client device and application on a server side. So, for the purpose of application-to-middleware communication performance, the size of GUI descriptions should be as small as possible.
- To enable flexible layouts, position and size specification should support not only fixed values but also mathematical expressions with references to position and size parameters of other GUI components. This guideline is a direct lesson from our experiment. Descriptions of GUI components that use fixed values may provide a sub-optimal or even unusable final GUI on some devices - e.g., devices with very large or very small screens. Also, fixed values make it difficult to reuse GUI components within different containers. With mathematical expressions instead of fixed values, it is possible to relate component dimensions to dimensions of its parent container or other components on the screen, which helps to avoid the mentioned problems.
- Another lesson learned from the presented experiment is related to the perceived designer workload while building a UI. According to experiences gathered from the development of test UI views, the job was easier with UIDLs that have low number of unique elements and attributes. This correlates with values of the clarity measure reported in Sect. 4. Therefore, to allow GUI designers to work with compact and clear descriptions, the total number of elements and attributes in the description documents and the number of unique elements should be as low as possible.
- Additionally, based on experiences from the experiment, it can be assumed that to provide the highest possible expressive power (in terms of GUI design) the UIDL should employ low-level graphical primitives, instead of high-level abstract GUI components.

The last guideline is an assumption that have been derived from lessons learned from the experiment. However, it should be noted, that despite the fact that informal observations allow formulating this guideline, it should be treated as a hypothesis that need further evaluation. Such evaluation will be carried out after formulation of a UIDL optimized for DIA-based applications.

6 Conclusions

This paper presents results of an experiment in which eight popular UIDLs have been used to build device-independent descriptions of GUI views, reflecting requirements of a practical mobile application. The descriptions used to express *composed GUI views* and *individual GUI components* have been assessed using a set of qualitative and quantitative criteria.

The gathered results confirm the formulated research hypothesis that selected popular declarative UIDLs are not capable of fully describing mobile GUI views in a device-independent manner for DIA-based applications, where the GUI description is supposed to be unambiguously and automatically processed by a GUI adaptation middleware. None of the analyzed UIDLs can be directly used for DIA-based applications that employ a GUI adaptation middleware to automatically generate a device-specific version of application GUI. Therefore, there is a need to either develop a new UIDL adopting the best solutions, or extend one of the best suited UIDLs. Such new solution should follow the set of guidelines inferred from lessons learned during the implementation of test GUI views and from the evaluation results.

As a follow-up to the presented analysis, we plan to prepare a prototype of an optimal device-independent UIDL for DIA-based applications and to evaluate it on a practical use case.

References

1. Abrams M, Phanouriou C, Batongbacal AL, Williams SM, Shuster JE (1999) UIML: an appliance-independent XML user interface language. Comput Netw 31(11):1695–1708
2. Android XML (2014) http://developer.android.com/guide/topics/ui/index.html
3. Barry P (2008) Introducing OpenLaszlo 4. Linux J 2008(171):4
4. BZWBK mobile application (2014) https://play.google.com/store/apps/details?id=pl.bzwbk.bzwbk24
5. Calvary G, Coutaz J, Thevenin D, Limbourg Q, Bouillon L, Vanderdonckt J (2003) A unifying reference framework for multi-target user interfaces. Interact Comput 15(3):289–308. doi:10.1016/S0953-5438(03)00010-9
6. Chmielewski J (2013) Towards an architecture for future internet applications. In: The future internet. Springer, Berlin Heidelberg, pp 214–219. doi:10.1007/978-3-642-38082-2_18
7. Chmielewski J (2014) Device-independent architecture for ubiquitous applications. Pers Ubiquitous Comput 18(2):481–488. doi:10.1007/s00779-013-0666-y
8. Cisco: The Internet of things. How the next evolution of the Internet is changing everything (2011). https://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf
9. Cisco: VNI forecast highlights (2015). http://www.cisco.com/web/solutions/sp/vni/vni_forecast_highlights/index.html
10. Coenraets C (2004) An overview of MXML. The Flex markup language. Adobe Systems, New York
11. Dalal M, Ghoda A (2011) XAML developer reference. Microsoft Press, New York

12. Guerrero-Garcia J, Gonzalez-Calleros JM, Vanderdonckt J, Munoz-Arteaga J (2009) A theoretical survey of user interface description languages: Preliminary results. In: Proceedings of the 2009 Latin American Web Congress (La-web 2009), LA-WEB '09, IEEE Computer Society, Washington, DC, USA, pp 36–43. doi:10.1109/LA-WEB.2009.40
13. HTML5 (2013) http://www.w3.org/TR/html5/
14. Johanson B, Hutchins G, Winograd T, Stone M (2002) PointRight: experience with flexible input redirection in interactive workspaces. In: Proceedings of the 15th annual ACM symposium on user interface software and technology, UIST'02, ACM, New York, NY, USA, pp 227–234. doi:10.1145/571985.572019
15. Limbourg Q, Vanderdonckt J, Michotte B, Bouillon L, López-Jaquero V (2004) UsiXML: a language supporting multi-path development of user interfaces. EHCI/DS-VIS 3425:200–220
16. McFarlane N (2004) Rapid application development with Mozilla. Prentice Hall Professional, New York
17. Model-based user interfaces (MBUI) Working Group (2014) http://www.w3.org/TR/2014/NOTE-mbui-intro-20140107/
18. Paternò F (2003) Models for universal usability. In: Proceedings of the 15th French-speaking conference on human-computer interaction on 15eme Conference Francophone sur l'Interaction Homme-Machine, ACM, pp 9–16
19. Paternò F (2005) Model-based tools for pervasive usability. Interact Comput 17(3):291–315
20. Paterno F (2013) User interface design adaptation. In: Soegaard M, Dam RF (eds) The Encyclopedia of human-computer interaction, 2nd edn. Interaction Design Foundation. https://www.interaction-design.org/literature/book/theencyclopedia-of-human-computer-interaction-2nd-ed
21. Paterno F, Santoro C, Spano LD (2009) MARIA: a universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. ACM Trans Comput Hum Interact (TOCHI) 16(4):19
22. Pohja M (2010) Comparison of common XML-based web user interface languages. J Web Eng 9(2):95–115
23. Rumiński D, Walczak K, Chmielewski J (2014) Generating user interfaces for XML Schema documents with a presentation language. In: Computer networks communications in computer and information science, vol 431. Springer, pp 328–337. doi:10.1007/978-3-319-07941-7_33
24. Silva CE, Campos JC (2012) Can GUI implementation markup languages be used for modelling? In: Proceedings of the 4th international conference on human-centered software engineering, HCSE'12, Springer-Verlag, Berlin, Heidelberg, pp 112–129. doi:10.1007/978-3-642-34347-6_7
25. Souchon N, Vanderdonckt J (2003) A review of XML-compliant user interface description languages. In: Interactive systems. design, specification, and verification, Springer, pp 377–391
26. Tacy A, Hanson R, Essington J, Tökke A (2013) GWT in action. Manning Publications, Greenwich
27. Thelin J (2011) Quick user interfaces with Qt. Linux J 2011(204):7
28. Trewin S, Zimmermann G, Vanderheiden G (2003) Abstract user interface representations: How well do they support universal access? In: Proceedings of the 2003 conference on universal usability, CUU'03, ACM, New York, NY, USA, pp 77–84. doi:10.1145/957205.957219
29. Vanderdonckt J, Calvary G, Coutaz J, Stanciulescu A (2008) Multimodality for plastic user interfaces: models, methods, and principles. In: Multimodal user interfaces. Springer, pp 61–84
30. Walczak K, Wiza W, Chmielewski J (2012) Adaptation of user interfaces in SOA applications. e-Minds. Int J Hum Comput Interact 2:3–17
31. Wolfram S (2014) Launching the Wolfram connected devices project, stephen wolfram blog. http://blog.stephenwolfram.com/2014/01/launching-the-wolfram-connected-devices-project/